(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 377 990 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89313400.7

(51) Int. Cl.5: G06F 9/38, G06F 7/48

(22) Date of filing: 20.12.89

(30) Priority: 13.01.89 US 297782

(43) Date of publication of application:
18.07.90 Bulletin 90/29

(84) Designated Contracting States:
DE FR GB

(71) Applicant: International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: Nguyenphu, Myhong
11802 Prairie Hen Lane
Austin Texas 78758(US)
Inventor: Thatcher, Larry Edward
11507 D-K Ranch Road
Austin Texas 78759(US)

(74) Representative: Grant, Iain Murray
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) Data processing systems.

(57) A data processing system including a first processor that performs fixed point arithmetic operations and a second processor that performs floating point arithmetic operations. These two processors are connected by control circuitry that decodes a floating point arithmetic instruction that requires the second processor to perform a specified floating point arithmetic operation. The control circuitry provides information to the first processor to enable the first processor to compute a memory address for accessing the floating point data required by the second processor for performing the specified floating point arithmetic operation. Simultaneously the control circuitry provides the second processor with the information to initiate the execution of the specified floating point arithmetic operation. Also, the data processing system includes the means to access multi-word floating point data on either even or odd memory address boundaries.

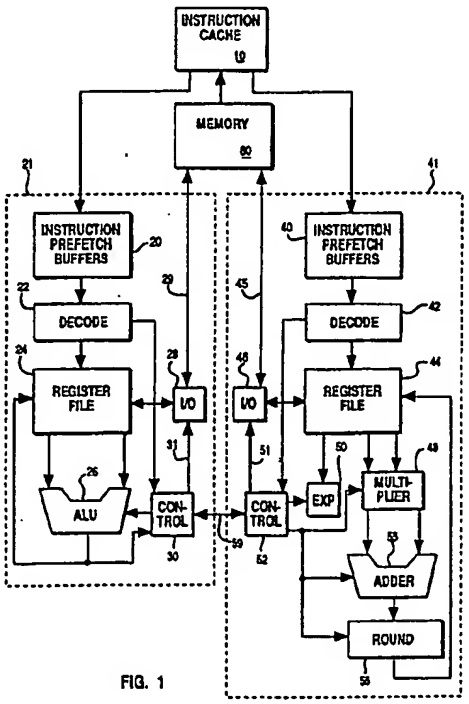

FIG. 1

EP 0 377 990 A2

DATA PROCESSING SYSTEMS

This invention relates to data processing systems using a multiple of processors to execute floating point arithmetic operations.

Traditionally floating point operations have been performed by a floating point processor. In some systems, such as the IBM RT workstation, the floating point processor functions as a peripheral unit relative to the central processing unit. The operation of this external floating point processor is not closely synchronised with the operation of the central processing unit.

Floating point operations often require data that consists of a multiple of data words. Normally, the use of interleaved memory requires that data words be accessed starting at even addresses. This enables the even and odd words to be transmitted by some systems in parallel. Therefore, the loading and storing of floating point data requires that the data be stored on even memory address boundaries. When floating point arithmetic data has been stored on odd memory address boundaries, a system interrupt has been traditionally generated so that the system software can compensate for the misalignment of the floating point arithmetic data to enable the floating point processor to perform the specified floating point operation.

Examples of the prior art that address misaligned data include

IBM Technical Disclosure Bulletin, Vol 26, No. 12, May, 1984, pages 6473-6475, entitled "Memory Transfer at Arbitrary Byte Boundaries" which discloses a mechanism for reordering data bytes for transfer.

IBM Technical Disclosure Bulletin, Vol. 27, No. 1A, June, 1984, pages 95-100, entitled "Dynamic Boundary Algorithm for a Data Movement Mechanism" discloses an algorithm to transfer data in groups having links specified by the address boundaries.

IBM Technical Disclosure Bulletin, Vol. 27, No. 3, August, 1984, pages 1585-1587, entitled "Storage Write Operation Control" discloses the performance of a storage write operation when different write capacities are provided, i.e., full word, half word, and character transfers with odd or even alignment.

IBM Technical Disclosure Bulletin, Vol. 24, No. 11B, April, 1982, pages 5948-5950, entitled "Odd-Byte Data Alignment Mechanism for System I/O" discloses an apparatus for achieving odd-byte data alignment for an I/O device.

It is the object of the present invention to provide a multiprocessing system for performing floating point operations. It is a further object of the present invention to provide a multiprocessing system for performing floating point operations wherein data stored for the floating point operations consist of multiple words and can be stored on either even or odd memory address boundaries.

The present invention provides a data processing system comprising:

a first processor means for performing fixed point arithmetic operations;

a second processor means for performing floating point arithmetic operations;

control means connected to the first and second processor means for decoding a floating point arithmetic instruction requiring the second processor means to perform a specified floating point arithmetic operation and providing information to the first processor means for computing a memory address for floating point data for the second processor means while providing information to the second processor means for execution of the specified floating point arithmetic operation.

As disclosed hereinafter by way of example, a data processing system is provided that consists of a first processor that performs a fixed point arithmetic operation and a second processor that performs floating point arithmetic operations. A control circuit is connected to the first processor and the second processor for decoding a floating point arithmetic instruction that requires the second processor to perform a specified floating point arithmetic operation. The control circuitry provides information to the first processor to enable the first processor to compute the memory address for the floating point data required for the second processor floating point operation while providing the floating point operation information to the second processor for the execution of the specified floating point arithmetic operation.

In other words, the first and second processors are connected by the control circuitry to enable the first processor to address floating point data in a memory for the second processor. This floating point data consists of a multiple of data words. According to the teachings of the invention the first of the data words stored in the memory can either be in an even or odd address location in memory. The control circuitry includes capability for determining whether an even or an odd address has been accessed for the first data word and for providing the proper storage of the data words in the second processor floating point registers.

The present invention will be described further by way of example with reference to an embodiment thereof as illustrated in the accompanying drawings, in which

Figure 1 is a block diagram of a multiproces-

sing system;

Figure 2 is a diagram illustrating the loading of two words in memory on an even memory address boundary;

Figure 3 is a block diagram illustrating the connection of the two floating point processor registers to the two word data buses;

Figure 4 is a diagram illustrating the loading of two data words on misaligned or odd memory address boundaries;

Figure 5 is a block diagram illustrating the connection of the floating point arithmetic registers to the two word data buses;

Figure 6 is a flow chart illustrating the control flow for the fixed point processor during a floating point load operation;

Figure 7 is a flow chart illustrating the control flow for the floating point processor during a floating point load operation;

Figure 8 is a timing diagram illustrating the pipeline operations of the fixed point and floating point processors executing a load operation on an even memory address boundary;

Figure 9 is a timing diagram illustrating the pipeline operation of the fixed point and floating point processors executing an arithmetic load operation on an odd or misaligned memory address boundary;

Figure 10 is a block diagram illustrating the connection of the two floating point processor registers to the two word data buses;

Figure 11 is a diagram illustrating the storage of two words in memory on an even memory address boundary;

Figure 12 is a block diagram illustrating the connection of the floating point arithmetic registers to the two word data buses;

Figure 13 is a diagram illustrating the storage of two data words on misaligned or odd memory address boundaries;

Figure 14 is a flow chart illustrating the control of the fixed point unit during a floating point arithmetic store operation;

Figure 15 is a flow chart illustrating the control flow the floating point processor during the performance of a floating point arithmetic store operation;

Figure 16 is a timing diagram illustrating the pipeline operation of the fixed point processor and the floating processor during the execution of a floating point arithmetic store operation on a even word boundary; and

Figure 17 is a timing diagram illustrating the pipeline operation of the fixed point processor and the floating point processor during the floating point arithmetic store operation on odd or misaligned memory address boundaries.

Figure 1 is a block diagram illustrating the interconnection of a fixed point processor 21 to a floating point processor 41. Both the fixed point processor 21 and the floating point processor 41 are connected to an instruction cache 10 and memory 80. Also the fixed point processor 21 is connected to the floating point processor 41 through control lines 59. Specifically the fixed point processor 21 control circuitry 30 is connected to the floating point processor 41 control circuitry 52 via lines 59. The fixed point processor consists of an instruction prefetch buffer 20 connected to the instruction cache 10. Instructions from the instruction prefetch buffer 20 are decoded in decoding circuitry 22. Data from the decoding circuitry 22 is provided to the register file 24. Also decoding information from the decoding circuitry 22 is provided to the control circuitry 30. The register file 24 is connected to the arithmetic logic unit (ALU) 26 and the I/O (input/output) circuitry 28. The I/O circuitry 28 is connected via line 29 to the memory 80. This I/O circuitry 28 is also connected to the register file 24 and to the control circuitry 30. The arithmetic logic unit is further connected to the control circuitry 30 and back to the register file 24 as shown.

The floating point arithmetic processor 41 also includes an instruction prefetch buffer 40 that is connected to a decode circuitry 42. The decode circuitry 42 is in turn connected to the control circuitry 52 and the register file 44. The register file 44 is connected to the I/O circuitry 46, the exponential circuitry 50, and a multiplier 48. The multiplier 48 is connected to an adder 55. The adder 55 is in turn connected to a rounding circuit 56. The control circuitry 52 is also connected to the multiplier 48, the adder 55, the rounding circuit 56 and I/O circuitry 46. Also the I/O circuitry 46 is connected to the memory 80 via line 45 and the register file 44. In operation, a floating point instruction that is stored in the instruction cache 10 is transferred to both the instruction prefetch buffer 20 in the fixed point processor 21 and the instruction prefetch buffer 40 in the floating point processor 41. In operation, the floating point instruction requires the fixed point processor 21 to calculate the address during the time that the floating point processor 41 initialises execution of the floating point instruction operation. The synchronisation of the fixed point processor 21 and the floating point processor 41 is accomplished by the two control units 30 and 52 communicating over lines 59. The specific floating point arithmetic operation performed is a floating point load from memory or floating point store to memory operation. Floating point data include a multiple of data words. In the disclosed arrangement, a floating point data word consists of at least two data words.

Figure 2 illustrates the storage of floating point

data consisting of two data words. In Figure 2 WORD 1 is stored on an even memory address boundary and WORD 2 is stored on an odd memory address boundary. The term address boundary relates to the value of the address itself. In other words, an even address boundary means that the memory address is an even number. Likewise, an odd address boundary means that the memory address is an odd number. The two word floating point data of Figure 2 is referred to as double word aligned data. The storage of the two words (WORD 1 and WORD 2) is illustrated in Figure 3. In Figure 3 a load register 110 is provided for the loading of the even word and register 112 is provided for the odd word. Multiplexers 122 and 124 are provided to switch the locations of the data stored in registers 110 and 112 if required. However, in this example, the double word data is aligned. Therefore, no changing of positions is required. The data from register 110 will be loaded through multiplexer 122 into register location 126. Likewise the data in register 112 will be loaded through multiplexer 124 into register 128. Registers 126 and 128 are the actual floating point processor unit internal registers.

Figure 4 illustrates the storage of a floating point double word on misaligned or odd word boundaries. In other words, in Figure 4, the first word of the data (WORD 2) is stored on an odd boundary and the second word (WORD 3) is stored on an even boundary. In Figure 5, WORD 3 is loaded into register 132 and WORD 2 is loaded into register 134. The contents of register 132 must be loaded into the floating point register 152. Likewise the contents of register 134 must be loaded into floating point register 150. This is accomplished by the multiplexers 146 and 148. Multiplexer 146 is connected to both register 132 and 134. Likewise, multiplexer 148 is connected to register 132 via lines 136 and 138. Therefore, under the control of control circuitry 30 and control circuitry 52, the multiplexers are configured such that the data from register 134 is loaded into register 150 and the data from register 132 is loaded into register 152.

Figure 6 illustrates the control flow of the control circuitry 30 in performing the address generation for a floating point load instruction. In step 160, the instruction received from the instruction cache 10 through the instruction prefetch buffer 20 is decoded in decoding circuitry 22. In step 162, the operation that is required from the decoded instruction is then executed. In step 164, the actual address is generated. In step 166, the control circuitry 30 checks the alignment of the generated address to determine if the address boundary is even and then aligned or odd and then misaligned. If the alignment is even, the control proceeds to step 168 to read the double word from memory and then to step 170 to provide the signal to the load register latches 132 and 134 (Figure 5) or registers 110 and 112 (Figure 3) to load its respective words from the memory. Then, the control proceeds to step 182 to signal the floating point processor that the data is ready. This signal is provided over one of lines 59. If the address generated is misaligned, i.e., the generated address is an odd address, the control proceeds to step 172 to read a single word from the odd address. Then in step 174, the odd latch (latch 134 in Figure 5 for example) is signalled to receive the word. The even address is then generated in step 176. In step 178, the even addressed word is then read. In step 180 the even register (register 132 in Figure 5 for example) is signalled to receive the even addressed word. The control then proceeds to step 182 as before. Upon completion of signalling that the data is ready the control flow returns to the beginning of step 160.

Figure 7 illustrates the control flow for the control circuitry 52 in the floating point arithmetic processor 41. In step 190, the instruction is decoded. Like the fixed point processor 21 the instruction for the floating point processor 41 originates from the instruction cache 10 through the instruction prefetch buffer 40 to the decoding circuitry 42. In step 192, the instruction operation dictated by the floating point instruction is executed. In step 194, the control circuitry 52 monitors the alignment control signals on line 59 from the control circuitry 30 of the fixed point processor 21. This monitoring determines whether or not the address that is being loaded begins at an even or an odd address. This decision is determined in 196 and if the load starts on an even address both the even and odd addressed words are loaded and received in step 198. Upon exiting step 198, the control determines in step 200 whether or not the data is then ready. If not, the control returns to step 194. If the data is ready, the control proceeds to step 210. Returning to step 196, if a load even signal is received from the control circuitry 30, the control proceeds to step 202 to receive the word on the even address. In step 204, the control determines whether the data is ready. If not, the control returns to step 194. If the data is ready, the control proceeds to step 206 to swap the even and odd words by using the multiplexers described in Figures 3 and 5. In step 196, after the decision has been made to receive the odd word, the odd word is received in step 208. In step 196 where an odd word is to be loaded first, the odd word is received first in step 208 before the even word is received in step 202. Therefore the data in the odd and even registers (132 and 134 of Figure 5 for example) are ready to be swapped. In step 210, the control circuitry 52 updates the registers in the floating

point register file 44. The control then returns to step 190 to determine the next instruction.

Figure 8 is a timing diagram illustrating the pipelined operation of the fixed point processor 21 (FXU) and the floating point processor 41 (FPU). Specifically the operation in Figure 8 is for an aligned access of data words. In cycle 1, the instruction (LOAD 1) is decoded by both the fixed point processor and the floating point processor. In cycle 2, the LOAD 1 instruction is used by the fixed point processor to generate the effective address and the odd and even words are loaded as shown. Also in cycle 2, the fixed point processor provides the ready signal. In cycle 2, the floating point processor receives the odd and even words together with executing the LOAD 1 instruction. In cycle 3, the floating point unit updates the floating point registers accordingly with the received data.

In Figure 9, a load of misaligned data is illustrated (i.e., data starting on an odd address boundary). In cycle 1, the LOAD 1 instruction is decoded by both processors. In cycle 2 the address for the load is generated in fixed point unit and the odd word (the first word of the two word set) is fetched. In the floating point unit in cycle 2, the LOAD 1 instruction is initially executed and the odd word is received by the odd register (134 in Figure 5). In cycle 3, the address for the even word is then generated and the floating point unit continues the execution of the instruction. Also in cycle 3, the even word is loaded from the fixed point unit and received by the floating point unit. The ready signal is provided in cycle 3 by the fixed point unit. In cycle 4, the floating point unit then performs the swap of the even and odd words into the appropriate floating point register file. The execution of a floating point arithmetic store operation is similar. In Figure 10, the block diagram illustrates the storage of the even and odd words from the registers onto the even data bus and odd data bus 316 and 318 respectively. Note that multiplexers 312 and 314 are provided such that the word stored in register 300 can be loaded on either the even data bus 316 or the odd data bus 318. Likewise the contents of register 302 (the odd register) can be loaded on either of the data buses 316 and 318. The illustration in Figure 11 shows the loading of an aligned word set having the first word (WORD 1) loaded in the even location and the second word loaded on the odd location. Figure 12 is a block diagram illustrating the loading of WORDS 2 and 3 on odd and even boundary respectively. In operation the contents of the even register 340 is initially loaded through multiplexer 354 onto the odd data bus line 358 by means of bus 346. Afterwards the contents of the odd register 342 is loaded over line 348 through multiplexer 352 onto the even data bus 356. The resulting storage appears in Figure 13 having the first word (WORD 2) stored on the odd boundary and the second word (WORD 3) stored on the even boundary as shown. The control flow for performing the floating point arithmetic store is illustrated in Figures 14 and 15. In Figure 14, the control flow for the fixed point unit is illustrated. Initially, the instruction is decoded in step 372 and executed in step 374. The address is generated in step 376. In step 378, the alignment is determined. If the alignment is even, the control proceeds to step 380 to enable the memory bus and the double word is written in step 382 (i.e., both the odd and even buses are used to write the odd and even words respectively to memory). The control then proceeds to step 396 to signal completion. If a non-even alignment is to be performed, the control proceeds to step 384 to enable the bus. Next in step 385, the swap command is issued causing the even store register to be place on the odd bus to memory as shown in Figure 12. Then in step 386, a write of the odd bus to memory is performed. In step 388, the even word address is generated. In step 390 again the bus is given the enable signal and in step 392 the odd store register is placed on the even data bus as previously discussed in Figure 12. In step 394 the write of the even bus to memory is performed. The control continues to step 396 to signal completion.

In Figure 15, the control flow in the floating point unit is illustrated. First the instruction is decoded in step 400 and then executed in step 402. Again the control lines 59 are monitored in step 404. In step 406, a determination is made as to whether the bus has been enabled. If not the control returns to step 404 to continue to monitor the bus. Once the bus has been enabled, the control flow proceeds to step 408 to determine if an odd/even swap is being signalled. If not, the control proceeds to step 412. If an odd/even swap is being signalled, the control flow proceeds to step 410 to signal the multiplexer (multiplexer 354 in Figure 12 for example) to perform the swap. In step 412, the control determines if the completion signal has been received. If not, the control returns to step 404. If the completion signal has been received, the control flow proceeds back to the start of the process.

Figure 16 is a timing diagram illustrating the operation of the fixed point processor (FXU) and the floating point processor (FPU) in a pipelined fashion while executing a floating point arithmetic store operation on an even word boundary aligned. Initially in cycle 1, both the fixed point processor and floating point processor decode the store 1 instruction. In cycle 2 the fixed point processor generates the effective address enables the bus while the floating point processor initiates execution of the store 1 instruction. In cycle 3 the fixed point

unit signals the storage of the even and odd words while the floating point unit in response to the signals places the even word on the even bus and odd word on the odd bus. Also, the completion signal is transferred.

Figure 17 is a timing diagram illustrating the operation of the fixed point processor and the floating point processor for an arithmetic store operation on an odd boundary (i.e., a non-even alignment or misalignment). In cycle 1 the store 1 instruction is decoded by both processors. In cycle 2 the fixed point processor generates the odd effective address and enables the bus while the floating point processor initiates execution. The fixed point unit issues a swap command to swap even/odd register connections to the even/odd buses. In cycle 3 the fixed point unit generates the effective address for the even data word, enables the bus again, issues a swap command and actually stores the odd bus. The floating point unit in response to the fixed point unit stores the odd word in the even register on the odd bus. In cycle 4 the even word storage is indicated by the fixed point unit and the even word in the odd register is placed on the even bus by the floating point unit. Then the fixed point unit signals the operation completion and the floating point unit receives this completion signal.

With such an arrangement, either even or odd alignment is executed avoiding the requirement of a software interrupt to "fix" alignment. This results in a higher performance in the speed of execution of floating point operations.

Claims

1. A data processing system comprising:
a first processor means for performing fixed point arithmetic operations;
a second processor means for performing floating point arithmetic operations;
control means connected to the first and second processor means for decoding a floating point arithmetic instruction requiring the second processor means to perform a specified floating point arithmetic operation and providing information to the first processor means for computing a memory address for floating point data for the second processor means while providing information to the second processor means for execution of the specified floating point arithmetic operation.

2. A system as claimed in Claim 1, wherein the first processor means includes means for fetching from a memory the floating point data at the memory address computed by the first processor means and for providing the floating point data.

3. A system as claimed in Claim 2, wherein the floating point data includes a plurality of data words.

4. A system as claimed in Claim 3, wherein the control means further includes means for loading the plurality of data words of the floating point data and wherein a first of the data words was stored in either an even or an odd address location of the memory.

5. A system as claimed in Claim 4, wherein the second processor means is connect to receive the floating point data on a data bus which includes an even data bus line for transferring data form even memory addresses and an odd data bus line for transferring data from odd memory address.

6. A system as claimed in Claim 5, wherein the second processor means includes loading means connected to the even data bus line, the odd data bus line and the control means for providing the first of the plurality of floating point data words to a first floating point register in the second processor means and a second of the plurality of the floating point data words to a second floating point register of the second processor means.

7. A system as claimed in Claim 6, wherein the second processor means includes a first multiplexor for loading a data word from either the odd data bus line or the even data bus line into the first floating point register of the second processor means.

8. A system as claimed in Claim 7, wherein the second processor means includes a second multiplexor for loading a data word from either the odd data bus line or the even data bus line into the second floating point register of the second processor means.

9. A system as claimed in Claim 8, wherein the second processor means first multiplexor includes means for loading data from the first floating point register on to either the even data bus line or the odd data bus line as specified by the control means.

10. A system as claimed in Claim 9, wherein the second processor means second multiplexor includes means for loading data from the second floating point register on to either the even data bus line or the odd data bus line as specified by the control means.

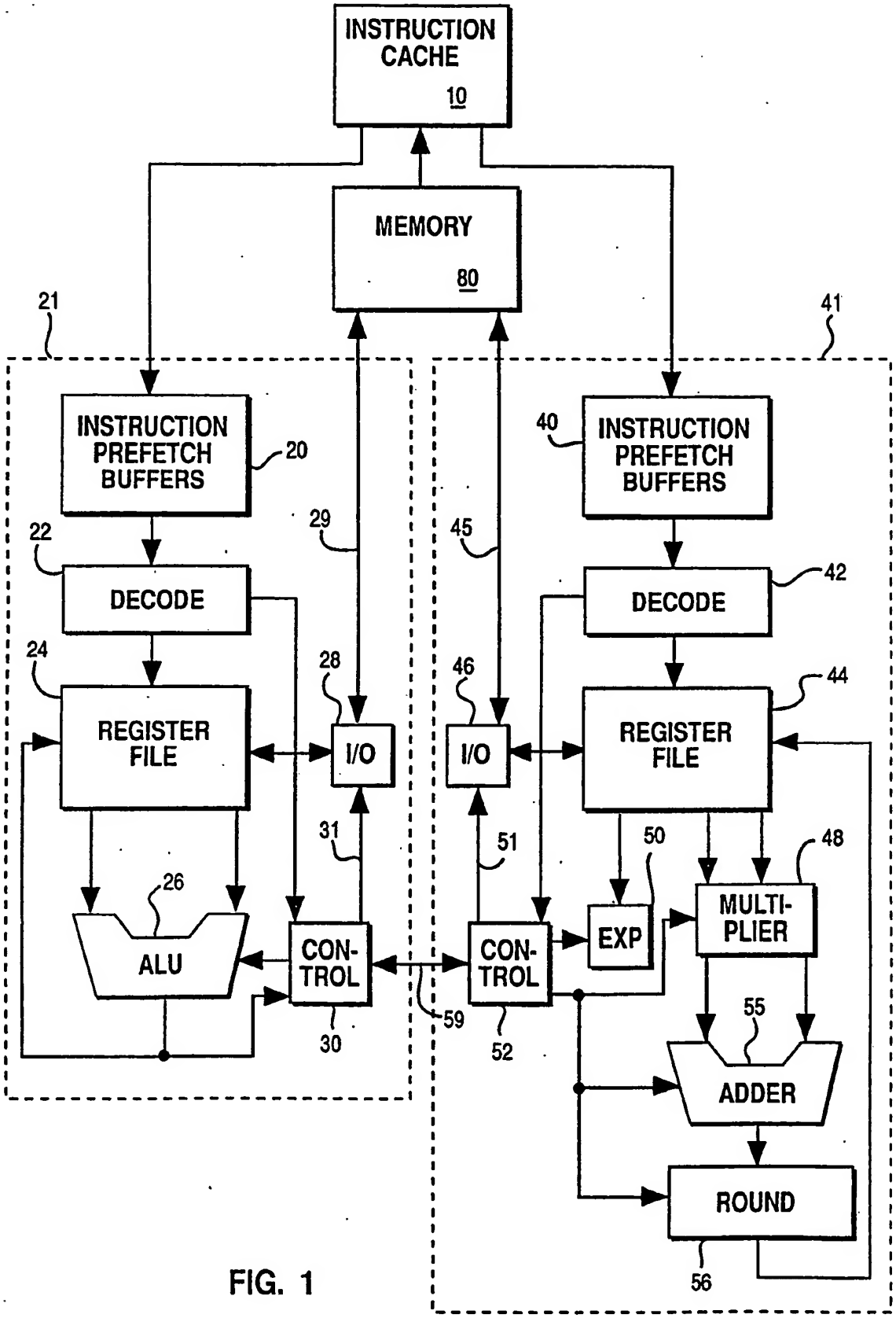
INSTRUCTION CACHE
10
MEMORY
80
21
41
INSTRUCTION PREFETCH BUFFERS
20
40
INSTRUCTION PREFETCH BUFFERS
22
29
45
DECODE
42
DECODE
24
28
46
44
REGISTER FILE
I/O
I/O
REGISTER FILE
31
51
50
48
26
MULTI-PLIER
EXP
ALU
CON-TROL
CON-TROL
59
52
55
ADDER
ROUND
30
56

FIG. 1

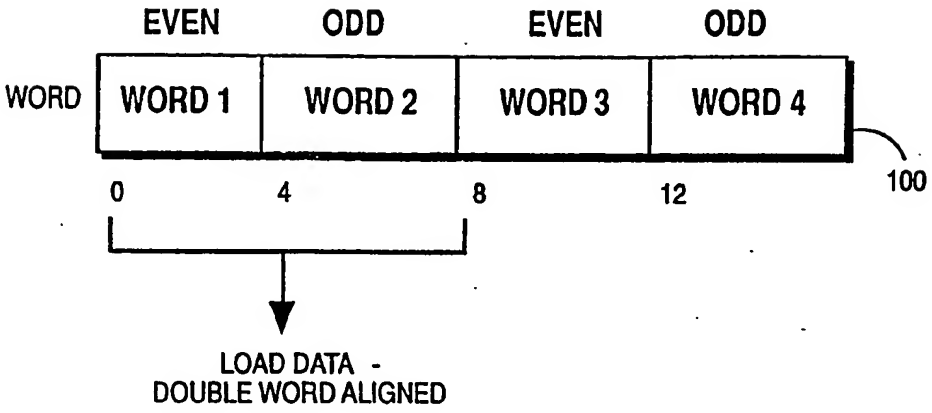
EVEN
ODD
EVEN
ODD
WORD
WORD 1
WORD 2
WORD 3
WORD 4
100
0
4
8
12
LOAD DATA -
DOUBLE WORD ALIGNED

FIG. 2

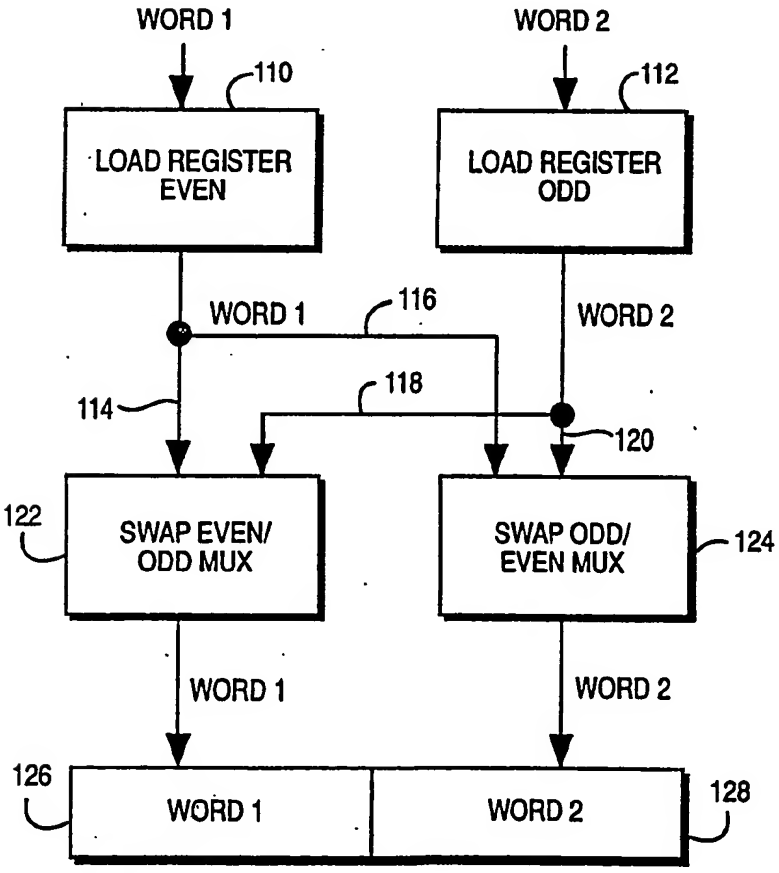
WORD 1
WORD 2
110
112
LOAD REGISTER EVEN
LOAD REGISTER ODD
WORD 1
116
WORD 2
114
118
120
122
SWAP EVEN/ ODD MUX
SWAP ODD/ EVEN MUX
124
WORD 1
WORD 2
126
WORD 1
WORD 2
128

FIG. 3

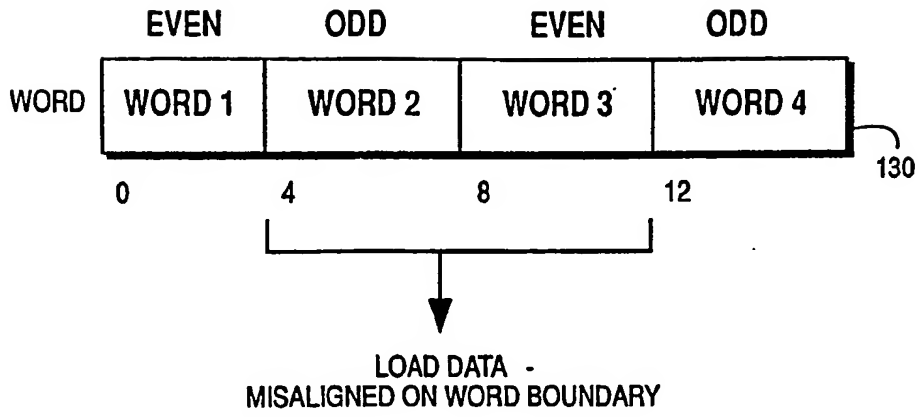
EVEN
ODD
EVEN
ODD
WORD
WORD 1
WORD 2
WORD 3
WORD 4
130
0
4
8
12
LOAD DATA -
MISALIGNED ON WORD BOUNDARY

FIG. 4

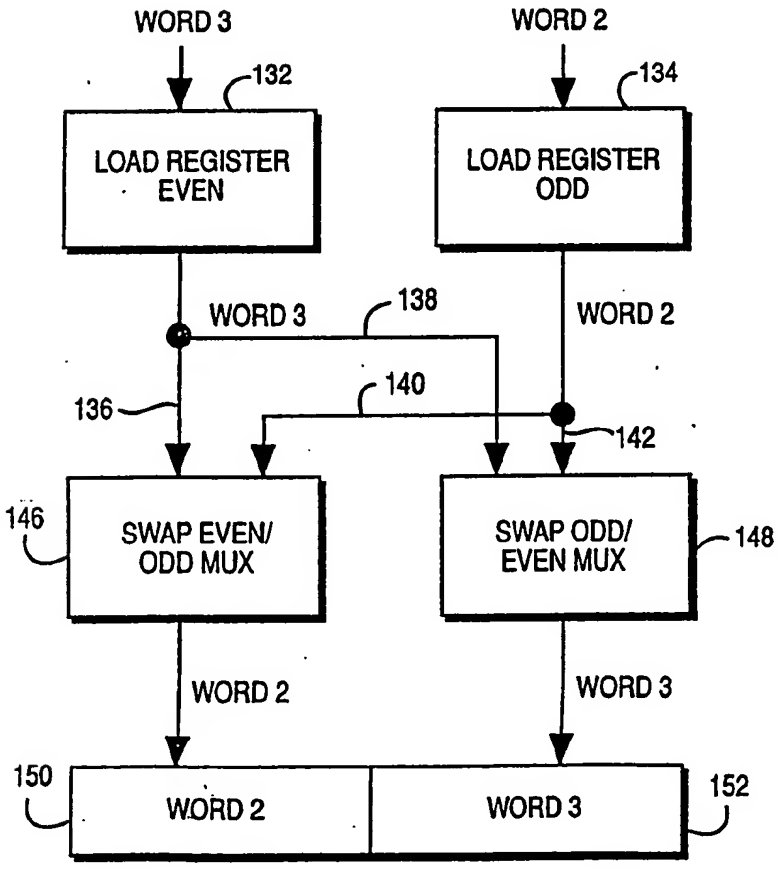
WORD 3
WORD 2
132
134
LOAD REGISTER EVEN
LOAD REGISTER ODD
WORD 3
138
WORD 2
140
136
142
146
SWAP EVEN/ ODD MUX
SWAP ODD/ EVEN MUX
148
WORD 2
WORD 3
150
WORD 2
WORD 3
152

FIG. 5

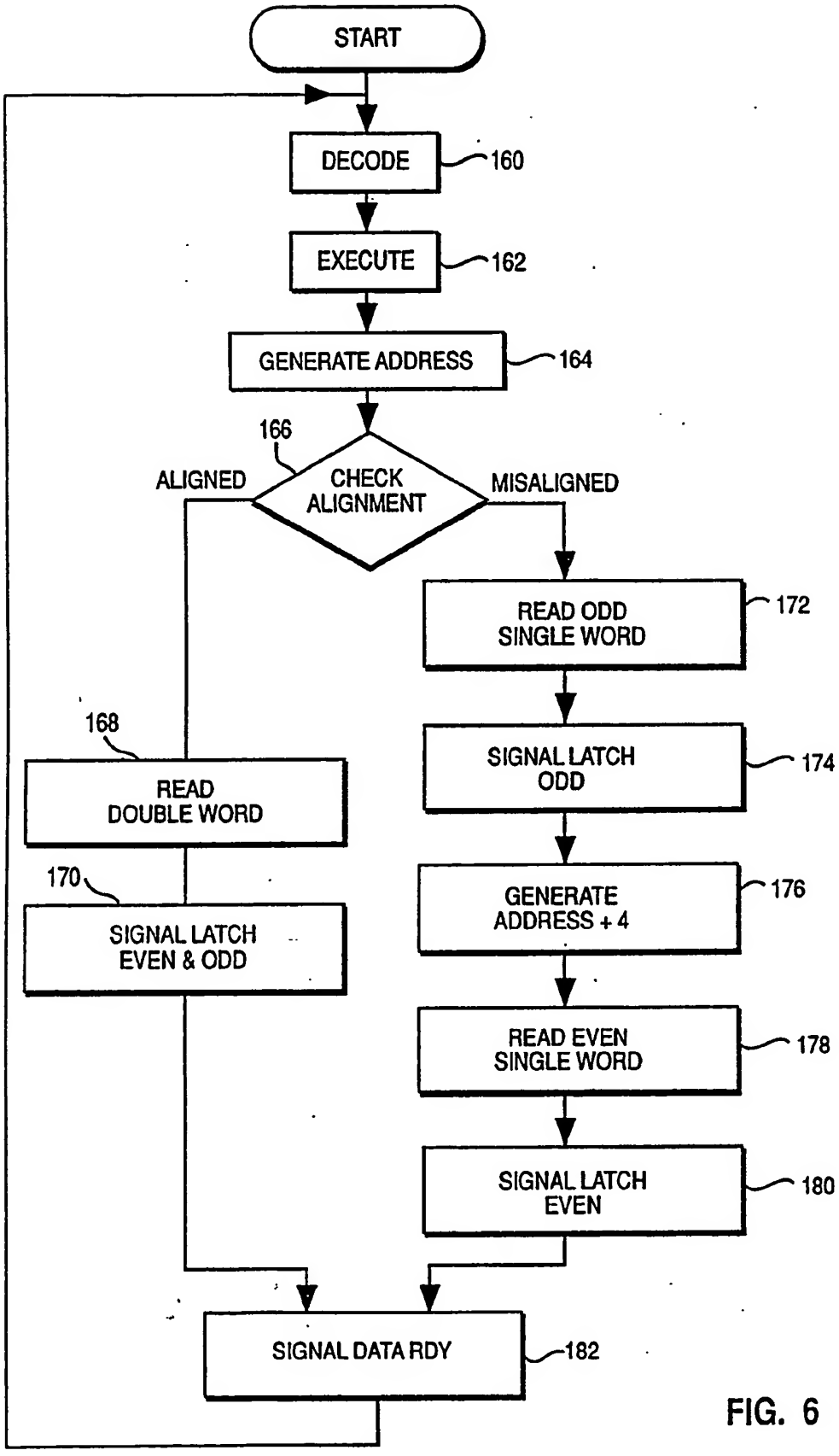
START
DECODE
160
EXECUTE
162
GENERATE ADDRESS
164
166
CHECK ALIGNMENT
ALIGNED
MISALIGNED
168
READ DOUBLE WORD
170
SIGNAL LATCH EVEN & ODD
READ ODD SINGLE WORD
172
SIGNAL LATCH ODD
174
GENERATE ADDRESS + 4
176
READ EVEN SINGLE WORD
178
SIGNAL LATCH EVEN
180
SIGNAL DATA RDY
182

FIG. 6

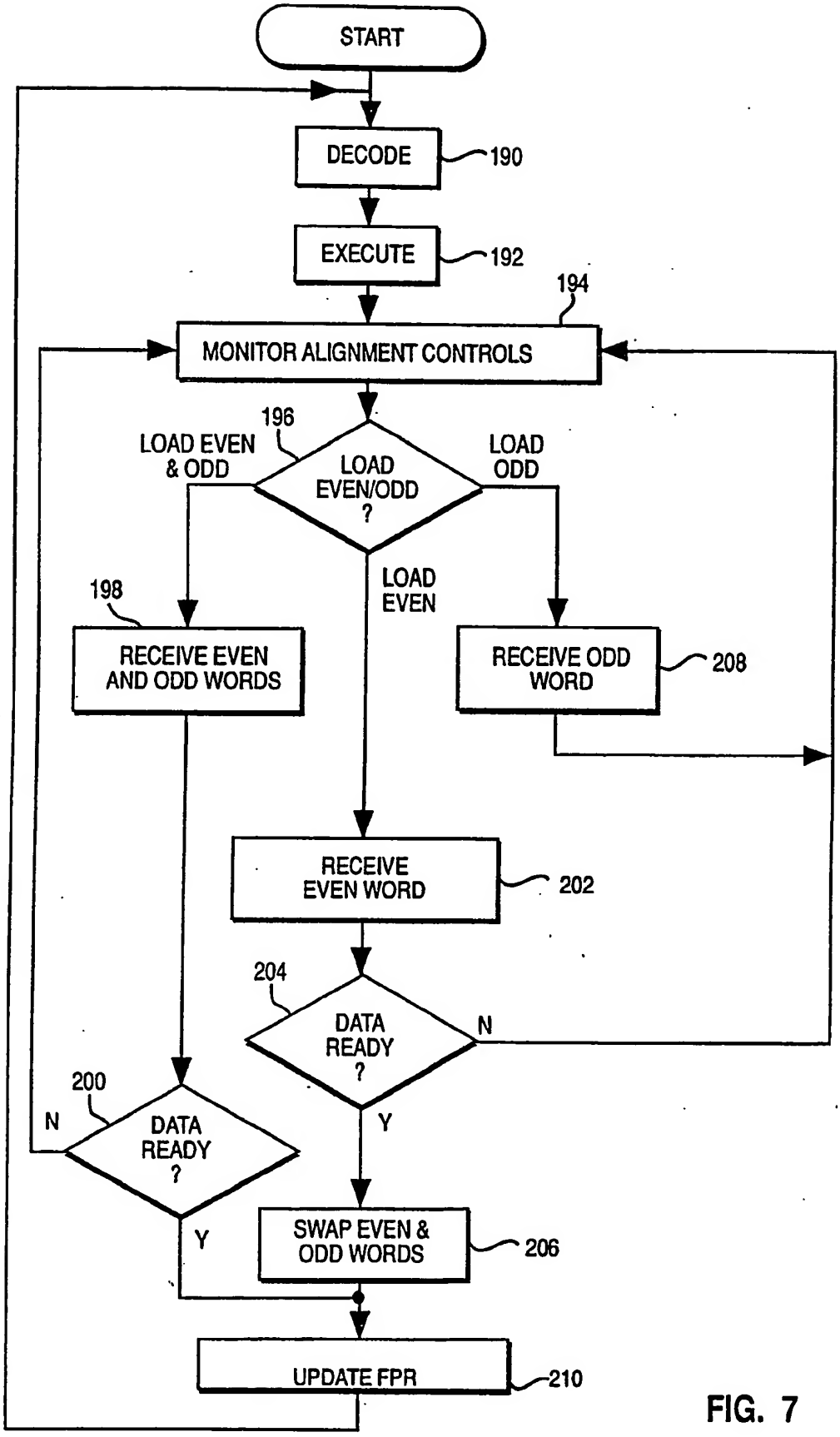
START
DECODE
190
EXECUTE
192
194
MONITOR ALIGNMENT CONTROLS
196
LOAD EVEN & ODD
LOAD EVEN/ODD ?
LOAD ODD
LOAD EVEN
198
RECEIVE EVEN AND ODD WORDS
RECEIVE ODD WORD
208
RECEIVE EVEN WORD
202
204
DATA READY ?
N
Y
200
N
DATA READY ?
Y
SWAP EVEN & ODD WORDS
206
UPDATE FPR
210

FIG. 7

CYCLE			1		2		3	
	FXU	FPU	FXU	FPU	FXU	FPU	FXU	FPU
FUNCTION	DECODE	DECODE	LOAD 1	LOAD 1				
	GENERATE E/A	EXECUTE			LOAD 1	LOAD 1		
	LOAD ODD	RECEIVE ODD			LOAD 1	LOAD 1		
	LOAD EVEN	RECEIVE EVEN			LOAD 1	LOAD 1		
	SIGNAL READY	SWAP EVEN/ODD			LOAD 1			
		UPDATE FPR						LOAD 1

FIG. 8

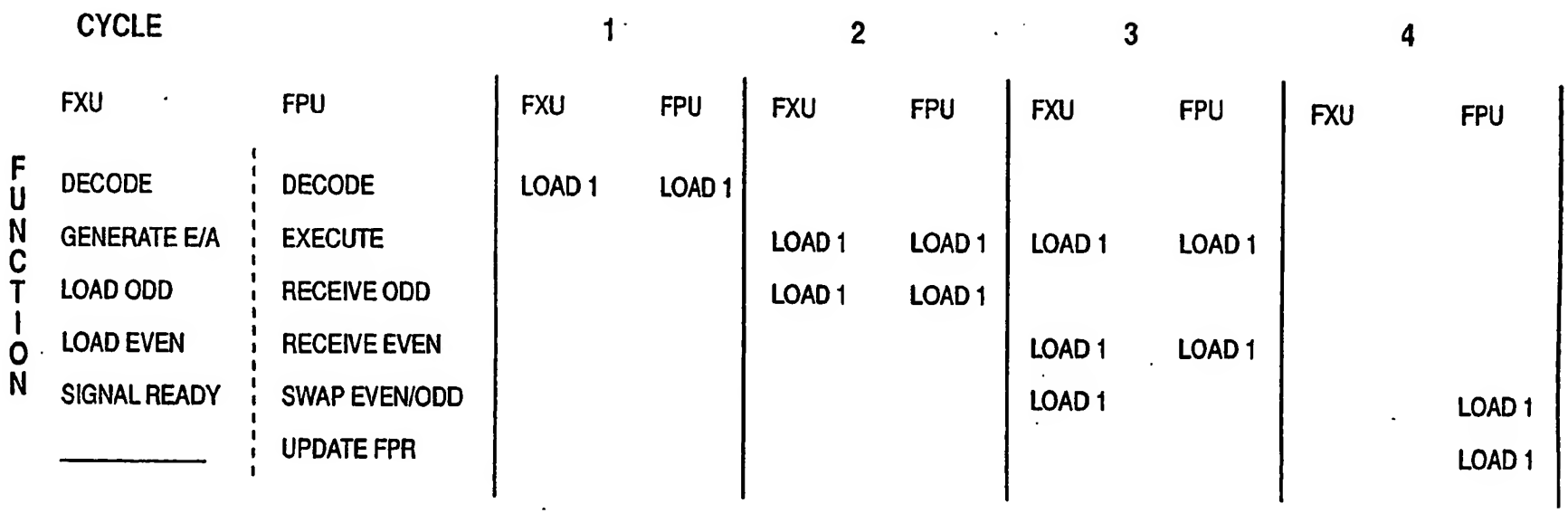

FIG. 9

EP 0 377 990 A2

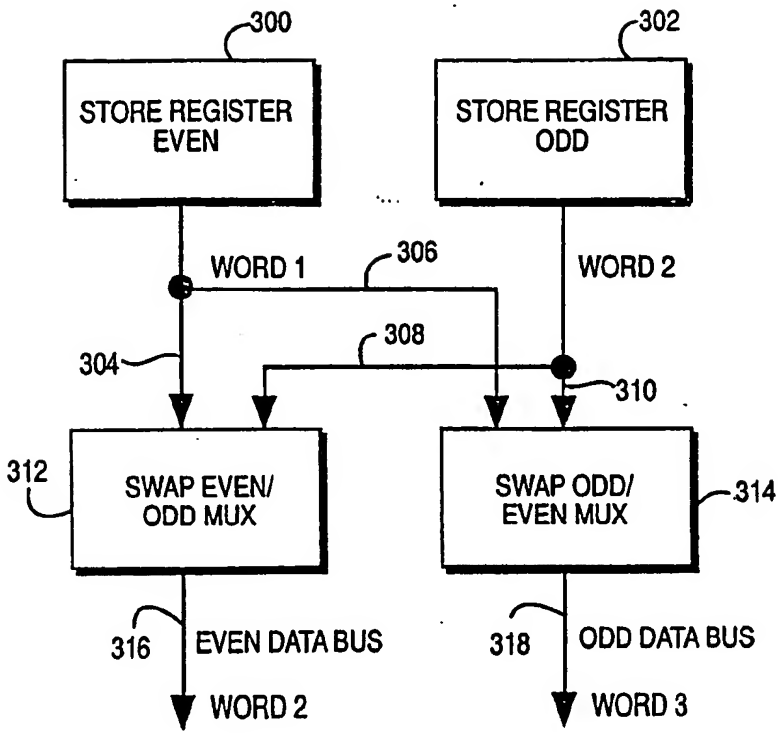
300
302
STORE REGISTER EVEN
STORE REGISTER ODD
WORD 1
306
WORD 2
308
304
310
312
SWAP EVEN/ ODD MUX
SWAP ODD/ EVEN MUX
314
316
EVEN DATA BUS
318
ODD DATA BUS
WORD 2
WORD 3

FIG. 10

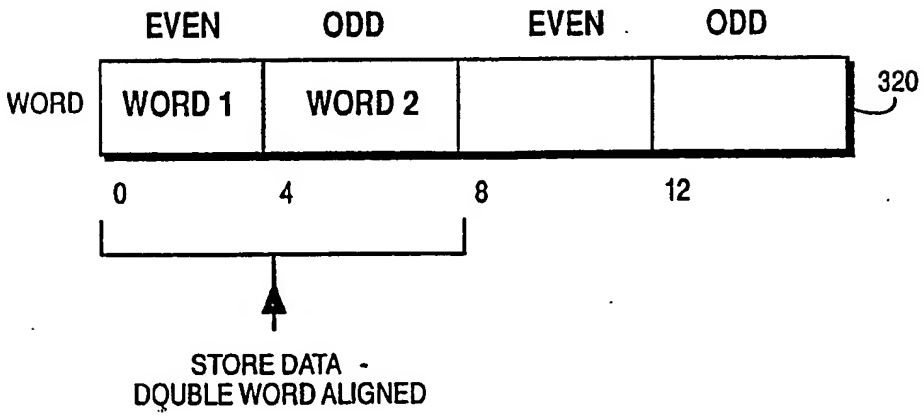
EVEN
ODD
EVEN
ODD
WORD
WORD 1
WORD 2
320
0
4
8
12
STORE DATA - DQUBLE WORD ALIGNED

FIG. 11

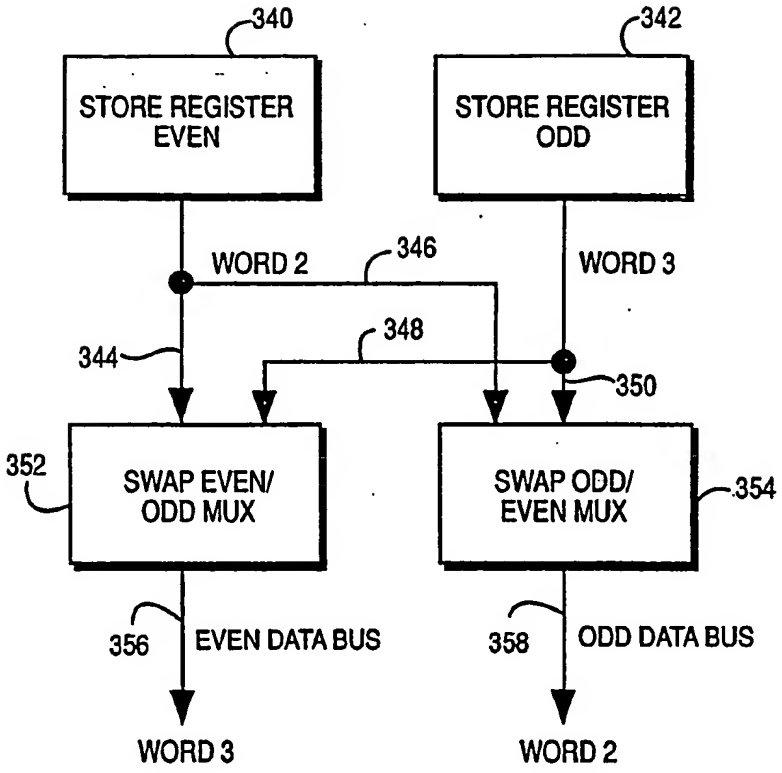
340
342
STORE REGISTER EVEN
STORE REGISTER ODD
WORD 2
346
WORD 3
348
344
350
352
SWAP EVEN/ ODD MUX
SWAP ODD/ EVEN MUX
354
356
EVEN DATA BUS
358
ODD DATA BUS
WORD 3
WORD 2

FIG. 12

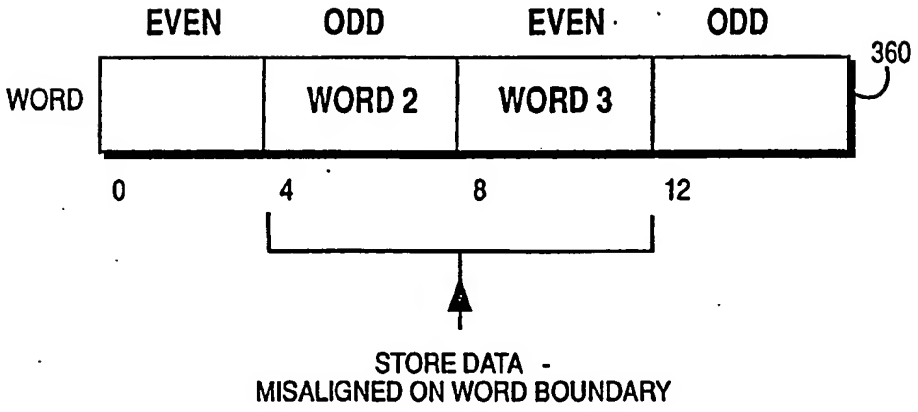
EVEN
ODD
EVEN
ODD
360
WORD
WORD 2
WORD 3
0
4
8
12
STORE DATA -
MISALIGNED ON WORD BOUNDARY

FIG. 13

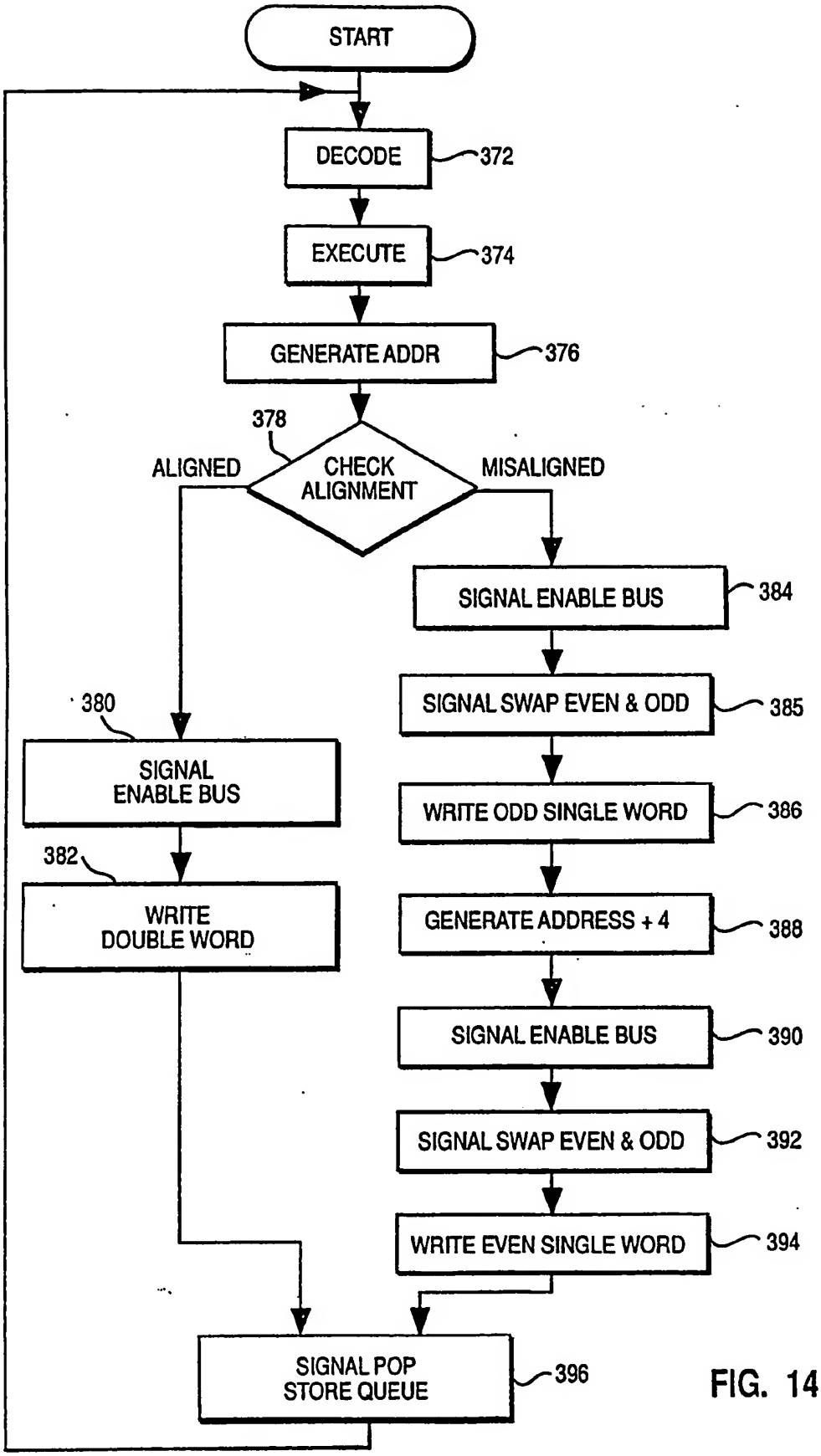
START
DECODE
372
EXECUTE
374
GENERATE ADDR
376
378
CHECK ALIGNMENT
ALIGNED
MISALIGNED
380
SIGNAL ENABLE BUS
382
WRITE DOUBLE WORD
SIGNAL ENABLE BUS
384
SIGNAL SWAP EVEN & ODD
385
WRITE ODD SINGLE WORD
386
GENERATE ADDRESS + 4
388
SIGNAL ENABLE BUS
390
SIGNAL SWAP EVEN & ODD
392
WRITE EVEN SINGLE WORD
394
SIGNAL POP STORE QUEUE
396

FIG. 14

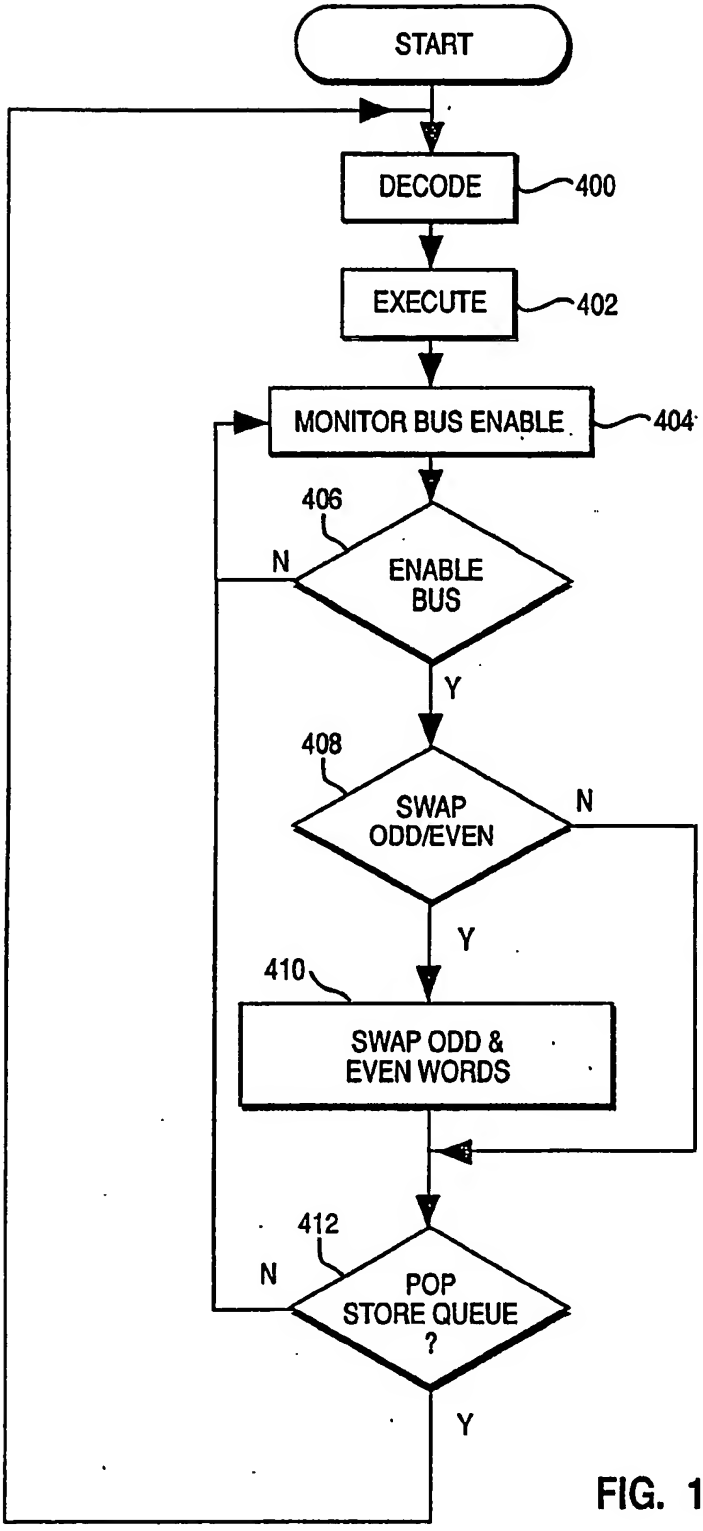
START
DECODE
400
EXECUTE
402
MONITOR BUS ENABLE
404
406
ENABLE BUS
N
Y
408
SWAP ODD/EVEN
N
Y
410
SWAP ODD & EVEN WORDS
412
POP STORE QUEUE ?
N
Y

FIG. 15

FPU STORE 1 - DOUBLE WORD ALIGNED

FUNCTION	CYCLE		1		2		3	
	FXU	FPU	FXU	FPU	FXU	FPU	FXU	FPU
	DECODE	DECODE	STORE 1	STORE 1				
	GENERATE E/A	EXECUTE			STORE 1	STORE 1		
	ENABLE BUS				STORE 1			
	STORE EVEN & ODD WORDS	PUT EVEN ON EVEN BUS PUT ODD ON ODD BUS					STORE 1	STORE 1
	SEND COMPLE-TION SIGNAL	RECEIVE COMPLETION SIGNAL					STORE 1	STORE 1

FIG. 16

EP 0 377 990 A2

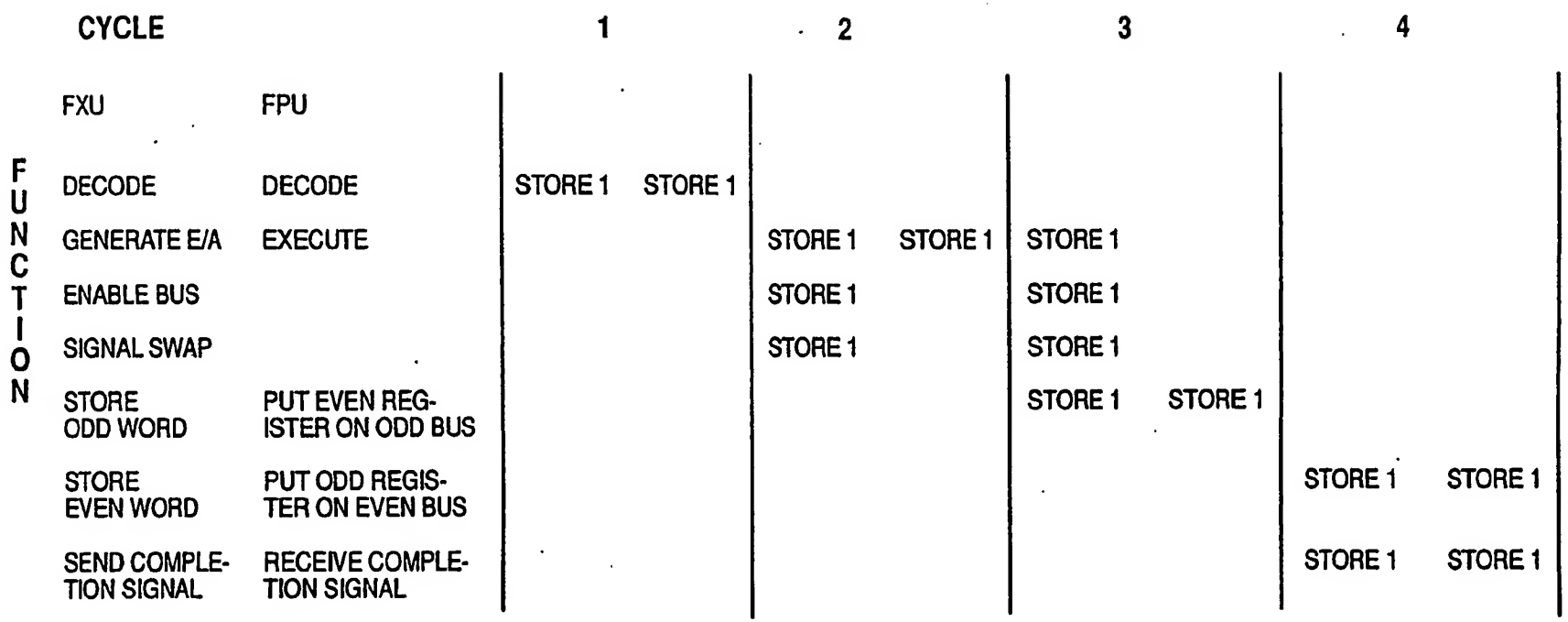

FIG. 17

EP 0 377 990 A2